Theoretical Computer Science 412 (2011) 753–764

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A new algorithm for aligning nested arc-annotated sequences under arbitrary weight schemes

Aïda Ouangraoua [a,*], Valentin Guignon [b], Sylvie Hamel [c], Cedric Chauve [d]

[a] INRIA LNE, LIFL, Université Lille 1, 40 avenue Halley, 59650, Villeneuve d'Ascq, France

[b] Bioversity International, CfL programme Parc Scientifique Agropolis II, 34397 Montpellier, France

[c] DIRO, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, Québec, H3C 3J7, Canada

[d] Department of Mathematics, Simon Fraser University, 8888 University Drive, Burnaby, BC, V5A 1S6, Canada

ARTICLE INFO

Article history:
Received 7 December 2008
Received in revised form 16 September 2010
Accepted 9 November 2010
Communicated by M. Crochemore

Keywords:
Arc-annotated sequences
Sequence alignment
Dynamic programming

ABSTRACT

In this paper, we propose a new algorithm for the alignment of nested arc-annotated sequences, having applications in the comparison of RNA secondary structures without pseudo-knots. We use a general edit distance model between arc-annotated sequences, that considers classical sequences of edit operations and structural edit operations on arcs. In this model, the general edit distance problem under a non-constrained weight scheme, is NP-hard. Recently, a hierarchy of arc-annotated sequence alignment problems that highlights less general, but tractable, problems was introduced. We refine this hierarchy of alignment problems and extend the class of tractable alignment problems. Up to date, the alignment problem we solve is the most general one that is known to be tractable in the considered edit distance model and under arbitrary weight schemes. This algorithm is efficient, as its asymptotic time and space complexities are the same as the complexities of the best previously published algorithm.

1. Introduction

An arc-annotated sequence is a sequence, over a given alphabet, with additional structure described by a set of arcs, each arc joining a pair of positions in the sequence. Arc-annotated have been widely studied, in particular due to the application in the representation of RNA secondary and tertiary structures [1–3,5,15,16,18]. RNA molecules, especially non-coding RNAs are indeed important molecules [6,21,24], whose function depends both on the sequence and on the structure. This motivates the need for efficient and accurate algorithms to compare RNA structures. In the present work, we consider the problem of computing an *edit distance* between *nested* arc-annotated sequences, that are commonly used to represent pseudoknot-free RNA secondary structures [13,14,21,23].

From a combinatorial point of view, nested arc-annotated sequences can be seen both as a generalization of sequences and as a special family of ordered trees. The problems of computing an edit distance between sequences or ordered trees using the three classical edit operations, insertion, deletion, and substitution are now well understood [7–9,12,17,20,25,26]. However, arc-annotated sequences, especially when used to model RNA secondary structures, can be compared using a larger set of edit operations that act on the arcs. In [15], new edit operations such as the creation, deletion or modification of arcs were introduced to account for structural evolutionary events on RNA molecules. Considering such operations naturally

* Corresponding author. Tel.: +33 3 57 59 79 74.
E-mail addresses: aida.ouangraoua@inria.fr, ouangaida@yahoo.fr (A. Ouangraoua), guignonv@yahoo.fr (V. Guignon), hamelsyl@iro.umontreal.ca (S. Hamel), cchauve@sfu.ca (C. Chauve).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.11.020

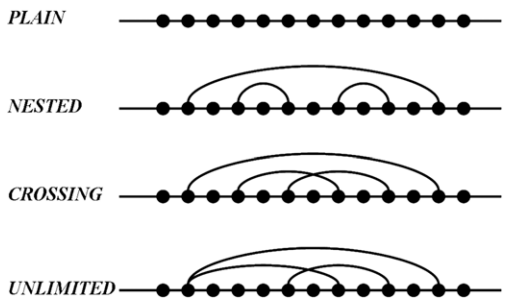

Fig. 1. Examples of arc-annotated sequences belonging to the classes of the hierarchy introduced in Definition 2: PLAIN, NESTED, CROSSING, UNLIMITED.

leads to more realistic alignments between RNA secondary structures [22], but at the weight of computational tractability. Indeed, it was recently shown in [4,19] that computing the edit distance between two nested arc-annotated structures in the model introduced in [15] is NP-hard. Several groups have defined less general comparison problems, by considering constraints either on the set of considered edit operations or on the weight scheme [13,24], or on the structure of possible alignments and edit sequences [3,5,11]. In particular, in [5], a hierarchy of several problems of edit and alignment distance computation between nested arc-annotated sequences was introduced, which enlightens the border between hard and tractable problems. Up to date, the most general tractable distance model, using the full set of edit operations introduced in [15] and under arbitrary weight schemes, was presented in [3].

The main contribution of our paper is to refine the hierarchy of arc-annotated sequence alignment problems defined in [3,5] and to introduce a new and more general alignment problem which is still tractable under arbitrary weight schemes. We propose an efficient dynamic programming algorithm for solving this new problem. In Section 2, we introduce some background on arc-annotated sequences and their comparison, including the hierarchy of arc-annotated sequence alignment problems introduced in [5], that we refine by introducing new classes. Finally, we introduce a new alignment problem which is tractable and can be solved with the same asymptotic complexity than the, less general, problem considered in [3] under the same conditions. In Section 3, we present a dynamic programming algorithm to solve this alignment problem. We conclude in Section 4.

2. Preliminaries: arc-annotated sequences and their comparison

We now describe formally nested arc-annotated sequences and different edit and alignment distance computation problems for the comparison of nested arc-annotated sequences with the existing results.

2.1. Arc-annotated sequences

Definition 1 (*Arc-annotated Sequence*)**.** An arc-annotated sequence of length n on a finite alphabet Σ is a couple $A = (S, P)$ where S is a sequence of length n on Σ and P is a set of pairs (i_1, i_2), with $1 \leq i_1 < i_2 \leq n$.

When arc-annotated sequences are used to represent RNA structures, $\Sigma = \{A, C, G, U\}$ is the alphabet of bases that compose an RNA molecule. Here we consider arbitrary alphabets, and we call an element of S a *base*. We denote by $S[i]$ the ith base of S and by $S[i_1..i_2]$ the sub-sequence of S containing bases $S[i_1], S[i_1 + 1], \ldots, S[i_2]$. A pair $(i_1, i_2) \in P$ represents an *arc* linking the base $S[i_1]$ called the *origin* of the arc and the base $S[i_2]$ called the *end* of the arc. An arc represents a *base pair* due to a chemical bond in terms of RNA structures. The bases $S[i_1]$ and $S[i_2]$ are then said to belong to the arc (i_1, i_2) and are the only bases that belong to this arc; a base that does not belong to any arc is called an *unpaired base*. We denote by $U(A)$, $P_o(A)$, and $P_e(A)$, respectively the set of unpaired bases, origins of arcs and ends of arcs of A. If $A = (S, P)$, we also use the notation $P(A) = P$ to denote the set of arcs of A. In an arc-annotated sequence, two arcs (i_1, i_2) and (i_3, i_4) are said to be *crossing*, if $i_1 < i_3 < i_2 < i_4$ or $i_3 < i_1 < i_4 < i_2$.

Arc-annotated sequences can be classified according to the combinatorial structure of their arcs. We now present the classification of arc-annotated sequences defined in [9] and used in [5] (Fig. 1).

Definition 2 (*Classification of Arc-annotated Sequences*)**.** An arc-annotated sequence $A = (S, P)$ is said to be (Fig. 1):

- UNLIMITED (UNLIM) if there is no restriction on P.
- CROSSING (CROS) if every base belongs to at most one arc.
- NESTED (abbreviated NEST) if it belongs to CROS but it has no pair of crossing arcs.
- PLAIN if P is empty.

2.2. Alignment of arc-annotated sequences

We consider the set of edit operations on arc-annotated sequences that was introduced in [15], defined below and illustrated in Figs. 2 and 3.

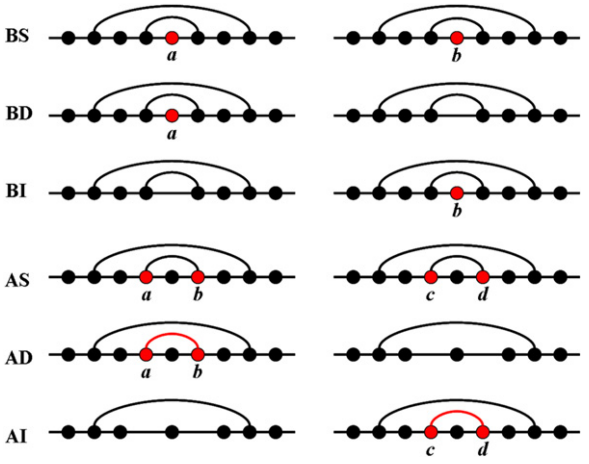

Fig. 2. Simple edit operations on arc-annotated sequences.

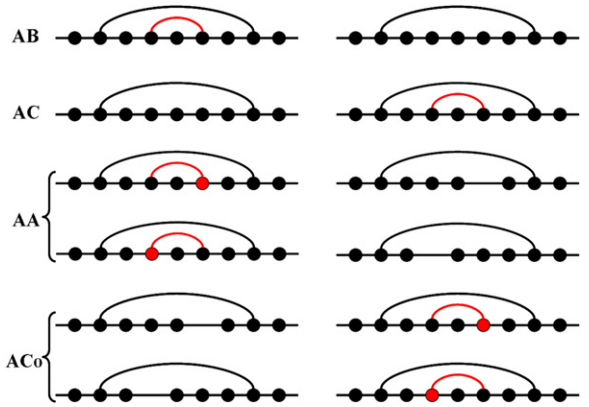

Fig. 3. Complex edit operations on arc-annotated sequences.

Definition 3 (*Edit Operations*)**.** Let a, b, c, d be elements (bases) of Σ.

- *The simple unpaired base edit operations are*:
  - Base-substitution (BS): substitution of a base a by a base b, denoted by $a \to b$.
  - Base-deletion (BD): deletion of a base a, denoted by $a \to \lambda$.
  - Base-insertion (BI): insertion of a base b, denoted by $\lambda \to b$.
- *The simple arc edit operations are*:
  - Arc-substitution (AS): substitution of an arc (a, b) by an arc (c, d), denoted by $(a, b) \to (c, d)$.
  - Arc-deletion (AD): deletion of an arc (a, b), denoted by $(a, b) \to \lambda, \lambda$.
  - Arc-insertion (AI): insertion of an arc (c, d), denoted by $\lambda, \lambda \to (c, d)$.
- *The complex arc edit operations are*:
  - Arc-breaking (AB): break of an arc (a, b), denoted by $(a, b) \to a, b$.
  - Arc-creation (AC): creation of an arc (c, d), denoted by $c, d \to (c, d)$.
  - Arc-altering (AA): alteration of an arc (a, b), denoted by $(a, b) \to a, \lambda$ or $(a, b) \to \lambda, b$.
  - Arc-completing (ACo): completion of an arc (c, d), denoted by $c, \lambda \to (c, d)$ or $\lambda, d \to (c, d)$.

Each edit operation e has a weight depending on the operation and on the bases it involves, denoted by $w(e)$. The set of weights associated with all possible edit operations is called the *weight scheme*. We naturally extend this notation as follows by identifying bases and their positions: for example, if i and j are the positions of two bases that form an arc in an arc-annotated sequence $A = (S, P)$, the weight of breaking this arc is denoted by $w((i, j) \to i, j)$, which is equivalent to $w((S[i], S[j]) \to S[i], S[j])$.

Definition 4 (*Edit Sequence and Distance*)**.** Let $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ be two arc-annotated sequences. An *edit sequence* between A_1 and A_2 is a sequence E of edit operations that transforms A_1 into A_2. The weight of an edit sequence E denoted by $w(E)$ is the sum of the weights of the edit operations that compose it. An edit sequence between two arc-annotated sequences is said to be *optimal* if its weight is minimal among all edit sequences between A_1 and A_2. The *edit distance* between A_1 and A_2 is the weight of an optimal edit sequence.

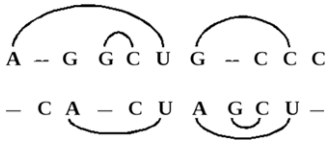

Fig. 4. An alignment between two nested arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ with $S_1 = AGGCUGCCC$, $P_1 = \{(1, 5), (3, 4), (6, 8)\}$, $S_2 = CACUAGCU$ and $P_2 = \{(2, 4), (5, 8), (6, 7)\}$. Following the usual convention, inserted or deleted bases are represented aligned with the symbol −.

Let C be a class of the hierarchy of arc-annotated structures. The problem of computing the edit distance between two arc-annotated sequences belonging to C is denoted by Edit(C, C).

Remark 5. A given weight scheme can implicitly prevent some edit operations to be considered in edit distance computation problems, if such operations can be replaced by a sequence of edit operations for a lesser weight. If such a situation does not occur, the weight scheme is said to be *complete*. In this paper, we consider arbitrary weight schemes, including complete weight schemes and non-complete weight schemes.

It was shown in [4,19] that Edit(Nest, Nest) is NP-hard, which motivated the introduction of restricted problems, based on the notion of alignment.

Definition 6 (*Alignment*)**.** An *alignment* between two arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ is a set $M = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ such that $1 \leq i_p < i_q \leq |S_1|$ and $1 \leq j_p < j_q \leq |S_2|$ for every $1 \leq p < q \leq k$. If $(i, j) \in M$, then $S_1[i]$ is said to be aligned with $S_2[j]$; we denote the fact that if $S_1[i]$ (resp. $S_2[j]$) is not aligned with any base of S_2 (resp. S_1) with $i \notin M$ (resp. $j \notin M$).

An alignment implicitly defines a set of edit operations and a weight, as follows: the notation below follows the names of the edit operations on arc-annotated sequences introduced in Definition 3 (see Fig. 4).

Definition 7 (*Weight of Alignment*)**.** Let M be an alignment between $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$.

$$\begin{array}{ll}
AS(M) & = \{((i_1, i_2), (j_1, j_2)) \in P_1 \times P_2 \mid (i_1, j_1) \in M \text{ and } (i_2, j_2) \in M\} \\
BS(M) & = \{(i, j) \in M \mid \nexists\, (i', j') \in M \text{ s.t. } ((i, i'), (j, j')) \in AS(M) \text{ or } ((i', i), (j', j)) \in AS(M)\} \\
BD(M) & = \{i \in U(A_1) \mid i \notin M\},\ BI(M) = \{j \in U(A_2) \mid j \notin M\} \\
AD(M) & = \{(i_1, i_2) \in P_1 \mid i_1 \notin M \text{ and } i_2 \notin M\},\ AI(M) = \{(j_1, j_2) \in P_2 \mid j_1 \notin M \text{ and } j_2 \notin M\} \\
AB(M) & = \{(i_1, i_2) \in P_1 \mid \exists\, (j_1, j_2) \notin P_2 \text{ s.t. } (i_1, j_1) \in M \text{ and } (i_2, j_2) \in M\} \\
AC(M) & = \{(j_1, j_2) \in P_2 \mid \exists\, (i_1, i_2) \notin P_1 \text{ s.t. } (i_1, j_1) \in M \text{ and } (i_2, j_2) \in M\} \\
AA_l(M) & = \{(i_1, i_2) \in P_1 \mid i_1 \notin M \text{ and } i_2 \in M\},\ AA_r(M) = \{(i_1, i_2) \in P_1 \mid i_1 \in M \text{ and } i_2 \notin M\} \\
ACo_l(M) & = \{(j_1, j_2) \in P_2 \mid j_1 \notin M \text{ and } j_2 \in M\},\ ACo_r(M) = \{(j_1, j_2) \in P_2 \mid j_1 \in M \text{ and } j_2 \notin M\}.
\end{array} \tag{1}$$

The weight of M, denoted by $w(M)$ is defined by

$$\begin{aligned}
w(M) = & \sum_{(i,j) \in BS(M)} w(i \to j) + \sum_{i \in BD(M)} w(i \to \lambda) + \sum_{j \in BI(M)} (\lambda \to j) + \sum_{((i_1,i_2),(j_1,j_2)) \in AS(M)} w((i_1, i_2) \to (j_1, j_2)) \\
& + \sum_{(i_1,i_2) \in AD(M)} w((i_1, i_2) \to \lambda, \lambda) + \sum_{(j_1,j_2) \in AI(M)} w(\lambda, \lambda \to (j_1, j_2)) \\
& + \sum_{(i_1,i_2) \in AB(M)} w((S_1[i_1], S_1[i_2]) \to S_1[i_1], S_1[i_2]) + \sum_{(j_1,j_2) \in AC(M)} w(S_2[j_1], S_2[j_2] \to (S_2[j_1], S_2[j_2])) \\
& + \sum_{(i_1,i_2) \in AA_l(M)} w((i_1, i_2) \to \lambda, i_2) + \sum_{(i_1,i_2) \in AA_r(M)} w(i_1, i_2) \to i_1, \lambda) \\
& + \sum_{(j_1,j_2) \in ACo_l(M)} w(\lambda, j_2 \to (j_1, j_2)) + \sum_{(j_1,j_2) \in ACo_r(M)} w(j_1, \lambda \to (j_1, j_2)).
\end{aligned} \tag{2}$$

For two arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$, an edit sequence E between A_1 and A_2 naturally induces an alignment M between A_1 and A_2 such that $w(M) \leq w(E)$. Conversely given an alignment between A_1 and A_2, the edit operations defined by M induce an edit sequence E between A_1 and A_2 such that $w(E) = w(M)$. Thus, the *edit distance* between two arc-annotated sequences is the minimum weight of an alignment between them.

2.3. A hierarchy of alignment problems

Definition 8 (*Super-sequence*)**.** A *super-sequence* of an arc-annotated sequence $A = (S, P)$ is an arc-annotated sequence which can be obtained by applying on A an edit sequence composed of insertion and substitution operations only: BI, AI, AC, ACo, BS, and AS; symmetrically, an arc-annotated sequence can be obtained from any of its super-sequences using only deletion and substitution operations.

Fig. 5. The super-sequence induced by the alignment of Fig. 4.

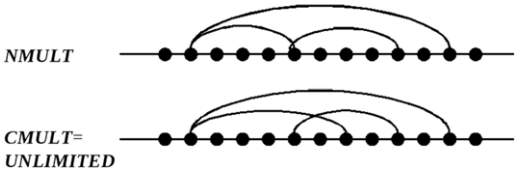

Fig. 6. Extension of the hierarchy of arc-annotated sequences.

An alignment M between two arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ naturally defines a super-sequence A_3 of A_1 and A_2 in the following way: the super-sequence A_3 is the arc-annotated sequence obtained by applying on A_1 a sequence composed of the set of edit operations E_1 defined by $BI(M), AI(M), AC(M), ACo_l(M)$ and $ACo_r(M)$ as introduced in Section 2.2. Since all these edit operations commute, the super-sequence A_3 does not depend on the order of the edit operations in the edit sequence E_1 (see Fig. 5).

Definition 9 (*Classification of Alignments and Alignment Distance*)**.** For a given class C of the hierarchy of arc-annotated sequences, an alignment M between two nested arc-annotated sequences A_1 and A_2 is said to be a C-alignment if the super-sequence induced by M belongs to C. We denote by $\mathcal{M}_C(A_1, A_2)$ the set of all C-alignments between A_1 and A_2. The alignment M is said to be an optimal C-alignment if its weight is minimal among the set of all C-alignment between A_1 and A_2. The weight of an optimal C-alignment between A_1 and A_2 is called the C-alignment distance between A_1 and A_2, denoted by $d_C(A_1, A_2)$: $d_C(A_1, A_2) = \min_{M \in \mathcal{M}_C(A_1,A_2)} w(M)$.

We denote by Align(Nest, Nest; C) the problem of computing the C-alignment distance between two nested arc-annotated sequences.

Theorem 10 (*[3,5]*)**.** *The problems* Edit(Nest, Nest) *and* Align(Nest, Nest; Unlim) *are equivalent.*

Theorem 10 immediately suggests a natural way to define restricted alignment problems in terms of the class of the allowed super-sequence. Using such an approach, some alignment problems were shown to be tractable, generalizing previous results on the alignment of arc-annotated sequences with simple operations, that were described in terms of alignment of trees [17].

Theorem 11 (*[3,5]*)**.** *Let A_1 and A_2 be two nested arc-annotated sequences of respective length n_1 and n_2. $d_{NEST}(A_1, A_2)$ can be computed in $O(n^4)$ worst-case time and $O(n^3)$ space where $n = n_1 + n_2$.*

Up to date, Align(Nest, Nest; Nest) is the most general problem that is known to be tractable with a complete weight scheme, as Align(Nest, Nest; Cros) has been shown to be NP-hard [4]. On the other hand there exist exact and polynomial time algorithms for alignment problems with super-sequence that are more general than nested, but where the range of considered edit sequences is restricted, as for example in [15,24] where AA, ACo, AI and AD are implicitly discarded, and in [13] where AB is the only considered complex arc operation.

2.4. Refining the hierarchy of arc-annotated sequences

We introduce here a new class of arc-annotated sequences. Our extension is inspired by the remark that there are two differences between Nest and Unlim: in Nest, (1) a base cannot belong to more than one arc and (2) arcs cannot be crossing. In class Cros, constraint (2) is relaxed, and then relaxing constraint (1) from class Cros gives class Unlim. It is then natural to consider an alternative path from Nest to Unlim, by first relaxing constraint (1), then constraint (2).

Definition 12 (Mult *Extension*)**.** Let C be a class of the hierarchy of arc-annotated sequences. We define the Mult extension of C, denoted by CMult by allowing bases to belong to more than one arc.

Hence, Unlim can in fact be seen as CrosMult, that is the MULT extension of Cros (Fig. 6). We denote NestedMult by NMult.

Property 13. Let C be a class of the hierarchy of arc-annotated sequences containing Nest, M an optimal C-alignment between two nested arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$, with a complete weight scheme. Let x and y be two bases of A_1 and A_2 respectively, such that x is aligned with y in M.

(1) In the super-sequence A_3 of A_1 and A_2 induced by M, x can belong to at most two arcs: one arc to which x belongs in A_1, and one arc to which y belongs in A_2.
(2) If C is either Nest or Cros (i.e. not a MULT extension), then x can belong to at most one arc in A_3.

Property 13 above follows in a straightforward way from the definition of classes of arc-annotated sequences (Definitions 2 and 12). It implies that, if NMult super-sequences are considered, a given base can be involved in two complex arc operations in an optimal alignment sequence, which is not the case if Cros-alignments are considered. This is the main interest of considering NMult-alignments.

3. An algorithm for Align(Nest, Nest; NMult)

We describe now an algorithm that solves ALIGN(NEST, NEST;NMULT), and proves our main result (Theorem 14). As far as we know, the only other alignment algorithm that considered an NMULT super-sequence did not consider a full set of edit operations and a complete weight scheme [13].

Theorem 14. *Given two nested arc-annotated sequences A_1 and A_2, of respective lengths n_1 and n_2, an optimal* NMULT*-alignment between A_1 and A_2 can be computed in $O(n^4)$ worst-case time and $O(n^3)$ space where $n = n_1 + n_2$.*

Similarly to other arc-annotated sequence comparison algorithms [11,15], we use the dynamic programming tables which are indexed by pairs of sub-sequences of the two considered arc-annotated sequences, called *indexing pairs*, which can be related to the hierarchy of arc-annotated sequences.

Definition 15 (*Indexing Pairs*)**.** Let $A = (S, P)$ be an arc-annotated sequence, with S of length n, and C be a class of the hierarchy of arc-annotated sequences.

(1) An ordered pair of integers $I = (x, y)$, with $1 \leq x \leq y \leq n$, is an *indexing pair of type* C *for* A if the arc-annotated sequence $A' = (S, P \cup \{(x, y)\})$ belongs to C. Its length is $y - x + 1$.
(2) The indexing pair $I = (x, y)$ of A defines an arc-annotated sub-sequence of A, denoted by $A^I = (S^I, P^I)$, obtained from A by deleting from S all the bases that do not belong to $S[x..y]$ and from P all the arcs that have at least one of their bases that does not belong to $S[x..y]$.
(3) The *empty indexing pair*, denoted by $\emptyset$ is such that $A^\emptyset$ is the empty arc-annotated sequence.
(4) The set of all indexing pairs of type C of A plus the empty indexing pair is denoted by $I_C(A)$.

Property 16. Let $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ be two nested arc-annotated sequences, C a class of the hierarchy of arc-annotated sequences and $I = (x, y)$ an indexing pair of A_1 that is not of type C. Any alignment M between A_1 and A_2 such that $(x, j_1) \in M$, $(y, j_2) \in M$ and $(j_1, j_2) \in P_2$ is not a C-alignment.

Property 16 suggests that to compute the C-alignment distance, we need to consider only indexing pairs of type C as a more general indexing pair (x, y) could lead to an alignment sequence that is out of class C if an arc is created between x and y.

From now on, we assume that all indexing pairs and all alignments we consider are of type NMULT; in particular we use $I(A)$ instead of $I_{NMULT}(A)$.

Before introducing the four dynamic programming tables we use in our algorithm, we describe why more than one table is required. First, in an optimal NMULT-alignment M, given an arc (i_1, i_2) of P_1 and an arc (j_1, j_2) of P_2, it is possible that $S_1[i_1]$ is aligned with $S_2[j_1]$ in M while $S_1[i_2]$ is not aligned with $S_2[j_2]$ in M; this cannot happen in an optimal CROS-alignment for example. This implies that when considering the first base x of an arc (x, y) either from P_1 or from P_2, even if it is aligned with the first base p of an arc (p, q) of the other arc-annotated sequence, it cannot be decided which edit operation will apply on (x, y) (resp. (p, q)), according to Definition 7 until y (resp. q) is considered, and we then need to remember the configuration of x and p (i.e. aligned or not). Four tables D_f, D_l and D_{fl} are used to record such partial alignment decisions taken on arcs. The tables D, D_f, D_l and D_{fl} are two-dimensional tables indexed by indexing pairs (I, J) defined as follows.

Definition 17 (*Dynamic Programming Tables*)**.** Let $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ be two nested arc-annotated sequences, $I \in I(A_1)$ and $J \in I(A_2)$.

- The cell $D[I, J]$ contains the optimal weight of an alignment between the arc-annotated sub-sequences A_1^I and A_2^J.
- Let $I = (x, y)$ and $J = (p, q)$. The cell $D_f[I, J]$ (resp. $D_l[I, J]$; $D_{fl}[I, J]$) contains the minimal weight of an alignment $M_{I,J}$ between the arc-annotated sub-sequences A_1^I and A_2^J such that $S_1[x]$ is aligned with $S_2[p]$ in $M_{I,J}$ (resp. $S_1[y]$ is aligned with $S_2[q]$ in $M_{I,J}$; $S_1[x]$ is aligned with $S_2[p]$ in $M_{I,J}$ and $S_1[y]$ is aligned with $S_2[q]$ in $M_{I,J}$).
- $D_f[I, J]$, $D_l[I, J]$, and $D_{fl}[I, J]$ are not defined if exactly one of the two indexing pairs I and J is the empty indexing pair $\emptyset$.
- $D_{fl}[I, J]$ is not defined if exactly one of the two indexing pairs I or J is of length 1.

We first describe how to fill the first values of the tables D, D_f, D_l, D_{fl}, using the value ∞ for the cells that are not defined.

Lemma 18 (*Initialization of Tables*)**.** *For every non-empty indexing pair $I \in I(A_1)$ and $J \in I(A_2)$*

$$D[I, \emptyset] = \sum_{i \in U(A_1^I)} w(i \to \lambda) + \sum_{(i_1, i_2) \in P(A_1^I)} w((i_1, i_2) \to \lambda, \lambda).$$

$$D[\emptyset, J] = \sum_{j \in U(A_2^J)} w(\lambda \to j) + \sum_{(j_1, j_2) \in P(A_2^J)} w(\lambda, \lambda \to (j_1, j_2)).$$

$$D_f[I, \emptyset] = D_f[\emptyset, J] = D_l[I, \emptyset] = D_l[\emptyset, J] = D_{fl}[I, \emptyset] = D_{fl}[\emptyset, J] = \infty.$$

Moreover,

$$D[\emptyset, \emptyset] = D_f[\emptyset, \emptyset] = D_l[\emptyset, \emptyset] = D_{fl}[\emptyset, \emptyset] = 0.$$

Proof. Direct consequence of Definition 15. □

$$
\begin{array}{lcl}
Q_1(I) & = & \{(I_1, I_2) \in I(A)^2 \mid I_1 + I_2 = I,\ \nexists (i_1, i_2) \in P(A^I) \mid i_1 \in I_1, i_2 \in I_2\} \\
Q_2(I) & = & \{(I_1, I_2) \in Q_1(I) \mid I_1 = (x, z), z \in P_e(A^I)\} \\
Q_3(I) & = & \{(I_1, I_2) \in Q_1(I) \mid I_2 \neq \emptyset\} \\
Q_4(I) & = & \{(I_1, I_2) \in I(A)^2 \mid I_1 + I_2 = I,\ \nexists (i_1, i_2) \in P(A^I)\backslash\{(x, y)\} \mid i_1 \in I_1, i_2 \in I_2\} \\
Q_5(I) & = & \{(I_1, I_2) \in Q_4(I) \mid I_1 = (x, z), z \in P_e(A^I)\} \\
\hline
BS_1 & = & w(x \to p) + D[(x+1, y), (p+1, q)] \\
BD_1 & = & w(x \to \lambda) + D[(x+1, y), (p, q)], \\
BI_1 & = & w(\lambda \to p) + D[(x, y), (p+1, q)] \\
AS_1 & = & w((x, z_1) \to (p, r_1)) + D[(x+1, z_1 - 1), (p+1, r_1 - 1)] + D[(z_1 + 1, y), (r_1 + 1, q)]\} \\
AD_1 & = & \min_{(J_1,J_2)\in Q_1(J)}\{w((x, z_1) \to \lambda, \lambda) + D[(x+1, z_1 - 1), J_1] + D[(z_1 + 1, y), J_2]\} \\
AI_1 & = & \min_{(I_1,I_2)\in Q_1(I)}\{w(\lambda, \lambda \to (p, r_1)) + D[I_1, (p+1, r_1 - 1)] + D[I_2, (r_1 + 1, q)]\} \\
AB_1 & = & \min_{(J_1,J_2)\in Q_1(J)}\{w((x, z_1) \to x, z_1) + D_f[(x, z_1 - 1), J_1)] + D_f[(z_1, y), J_2]\} \\
AB_2 & = & \min_{(J_1,J_2)\in Q_2(J)}\{D_{fl}[(x, z_1), J_1] + D[(z_1 + 1, y), J_2]\} \\
AC_1 & = & \min_{(I_1,I_2)\in Q_1(I)}\{w(p, r_1 \to (p, r_1)) + D_f[I_1, (p, r_1 - 1)] + D_f[I_2, (r_1, q)]\} \\
AC_2 & = & \min_{(I_1,I_2)\in Q_2(I)}\{D_{fl}[I_1, (p, r_1)] + D[I_2, (r_1 + 1, q)]\} \\
AA_1 & = & \min_{(J_1,J_2)\in Q_1(J)}\{w((x, z_1) \to x, \lambda) + D_f[(x, z_1 - 1), J_1] + D[(z_1 + 1, y), J_2]\} \\
AA_2 & = & \min_{(J_1,J_2)\in Q_1(J)}\{w((x, z_1) \to \lambda, z_1) + D[(x+1, z_1 - 1), J_1] + D_f[(z_1, y), J_2]\} \\
AA_3 & = & \min_{(J_1,J_2)\in Q_2(J)}\{w((x, z_1) \to \lambda, z_1) + D_l[(x+1, z_1), J_1] + D[(z_1 + 1, y), J_2]\} \\
ACo_1 & = & \min_{(I_1,I_2)\in Q_1(I)}\{w(p, \lambda \to (p, r_1)) + D_f[I_1, (p, r_1 - 1)] + D[I_2, (r_1 + 1, q)]\}\} \\
ACo_2 & = & \min_{(I_1,I_2)\in Q_1(I)}\{w(\lambda, r_1 \to (p, r_1)) + D[I_1, (p+1, r_1 - 1)] + D_f[I_2, (r_1, q)]\} \\
ACo_3 & = & \min_{(I_1,I_2)\in Q_2(I)}\{w(\lambda, r_1 \to (p, r_1)) + D_l[I_1, (p+1, r_1)] + D[I_2, (r_1 + 1, q)]\} \\
BS_2 & = & w(x \to p) + D_l[(x+1, y), (p+1, q)] \\
BD_2 & = & w(x \to \lambda) + D_l[(x+1, y), (p, q)] \\
BI_2 & = & w(\lambda \to p) + D_l[(x, y), (p+1, q)] \\
AS_2 & = & w((x, z_1) \to (p, r_1)) + D[(x+1, z_1 - 1), (p+1, r_1 - 1)] + D_l[(z_1 + 1, y), (r_1 + 1, q)] \\
AD_2 & = & \min_{(J_1,J_2)\in Q_3(J)}\{w((x, z_1) \to \lambda, \lambda) + D[(x+1, z_1 - 1), J_1] + D_l[(z_1 + 1, y), J_2]\} \\
AI_2 & = & \min_{(I_1,I_2)\in Q_3(I)}\{w(\lambda, \lambda \to (p, r_1)) + D[I_1, (p+1, r_1 - 1)] + D_l[I_2, (r_1 + 1, q)]\} \\
AB_3 & = & \min_{(J_1,J_2)\in Q_1(J)}\{w((x, z_1) \to x, z_1) + D_f[(x, z_1 - 1), J_1)] + D_{fl}[(z_1, y), J_2]\} \\
AB_4 & = & \min_{(J_1,J_2)\in Q_2(J)}\{D_{fl}[(x, z_1), J_1] + D_l[(z_1 + 1, y), J_2]\} \\
AC_3 & = & \min_{(I_1,I_2)\in Q_1(I)}\{w(p, r_1 \to (p, r_1)) + D_f[I_1, (p, r_1 - 1)] + D_{fl}[I_2, (r_1, q)]\} \\
AC_4 & = & \min_{(I_1,I_2)\in Q_2(I)}\{D_{fl}[I_1, (p, r_1)] + D_l[I_2, (r_1 + 1, q)]\} \\
AA_4 & = & \min_{(J_1,J_2)\in Q_3(J)}\{w((x, z_1) \to x, \lambda) + D_f[(x, z_1 - 1), J_1] + D_l[(z_1 + 1, y), J_2]\} \\
AA_5 & = & \min_{(J_1,J_2)\in Q_1(J)}\{w((x, z_1) \to \lambda, z_1) + D[(x+1, z_1 - 1), J_1] + D_{fl}[(z_1, y), J_2]\} \\
AA_6 & = & \min_{(J_1,J_2)\in Q_2(J)}\{w((x, z_1) \to \lambda, z_1) + D_l[(x+1, z_1), J_1] + D_l[(z_1 + 1, y), J_2]\} \\
ACo_4 & = & \min_{(I_1,I_2)\in Q_3(I)}\{w(p, \lambda \to (p, r_1)) + D_f[I_1, (p, r_1 - 1)] + D_l[I_2, (r_1 + 1, q)]\}\} \\
ACo_5 & = & \min_{(I_1,I_2)\in Q_1(I)}\{w(\lambda, r_1 \to (p, r_1)) + D[I_1, (p+1, r_1 - 1)] + D_{fl}[I_2, (r_1, q)]\} \\
ACo_6 & = & \min_{(I_1,I_2)\in Q_2(I)}\{w(\lambda, r_1 \to (p, r_1)) + D_l[I_1, (p+1, r_1)] + D_l[I_2, (r_1 + 1, q)]\} \\
ABAC_1 & = & \min_{(I_1,I_2)\in Q_4(I)}\{w((x, y) \to x, y) + w(p, r_1 \to (p, r_1)) + D_f[I_1, (p, r_1 - 1)] + D_{fl}[I_2, (r_1 + 1, q)]\} \\
ABAC_2 & = & \min_{(I_1,I_2)\in Q_5(I)}\{w((x, y) \to x, y) + D_{fl}[I_1, (p, r_1)] + D_l[I_2, (r_1 + 1, q)]\} \\
ACAB_1 & = & \min_{(J_1,J_2)\in Q_4(J)}\{w(p, q \to (p, q)) + w((x, z_1) \to x, z_1) + D_f[(x, z_1 - 1), J_1] + D_{fl}[(z_1, y), J_2]\} \\
ACAB_2 & = & \min_{(J_1,J_2)\in Q_5(J)}\{w((p, q) \to p, q) + D_{fl}[(x, z_1), J_1] + D_l[(z_1 + 1, y), J_2]\} \\
ABACo & = & \min_{(I_1,I_2)\in Q_4(I)}\{w((x, y) \to x, y) + w(p, \lambda \to (p, r_1)) + D_f[I_1, (p, r_1 - 1)] + D_l[I_2, (r_1 + 1, q)]\} \\
ACAA & = & \min_{(J_1,J_2)\in Q_4(J)}\{w(p, q \to (p, q)) + w((x, z_1) \to x, \lambda) + D_f[(x, z_1 - 1), J_1] + D_l[(z_1 + 1, y), J_2]\} \\
ABBS & = & w((x, y) \to x, y) + w(x \to p) + D_l[(x+1, y), (p+1, q)] \\
ACBS & = & w(p, q \to (p, q)) + w(x \to p) + D_l[(x+1, y), (p+1, q)]
\end{array}
$$

Fig. 7. Notations used in Lemmas 20–23.

Definition 19 (*Partition of Indexing Pair*)**.** Given an indexing pair $I = (x, y)$ of an arc-annotated sequence $A = (S, P)$, we say that two indexing pairs I_1 and I_2 partition I, denoted by $I_1 + I_2 = I$, if either one of them is $\emptyset$ and the other one is equal to I, or $I_1 = (x, z)$ and $I_2 = (z + 1, y)$ with $x \leq z \leq y - 1$.

In order to shorten the presentation of the equations, we use in Lemmas 20, 21, 22, 23 the following notations: $I = (x, y) \in I(A_1)$ and $J = (p, q) \in I(A_2)$ are two non-empty indexing pairs of two arc-annotated sequences A_1 and A_2; z_1 (resp. r_1) is such that $x \leq z_1 \leq y$ (resp. $p \leq r_1 \leq q$) and $(x, z_1) \in P(A_1)$ (resp. $(p, r_1) \in P(A_2)$). Notations used in these four lemmas are described in Fig. 7 and proofs are provided after the four lemmas are stated. We also assume that if an indexing pair $I = (a, b)$ is such that $a > b$ (resp. $a + 1 > b$; $a > b - 1$; $a + 1 > b - 1$), then (a, b) (resp. $(a + 1, b)$; $(a, b - 1)$; $(a + 1, b - 1)$) represents the empty indexing pair $\emptyset$.

Lemma 20 (*Filling up Table D*)**.** (1) *If $x \in P_o(A_1^I)$ and $p \in P_o(A_2^J)$, then*
$D[I, J] = \min\{AS_1, AB_1, AB_2, AC_1, AC_2, AA_1, AA_2, AA_3, ACo_1, ACo_2, ACo_3, AD_1, AI_1\}$.
(2) *If $x \in P_o(A_1^I)$ and $p \in U(A_2^J)$, then*
$D[I, J] = \min\{AB_1, AB_2, AA_1, AA_2, AA_3, AD_1, BI_1\}$.
(3) *If $x \in U(A_1^I)$ and $p \in P_o(A_2^J)$, then*
$D[I, J] = \min\{AC_1, AC_2, ACo_1, ACo_2, ACo_3, AI_1, BD_1\}$.
(4) *If $x \in U(A_1^I)$ and $p \in U(A_2^J)$, then* $D[I, J] = \min\{BS_1, BD_1, BI_1\}$.

Lemma 21 (*Filling up Table D_f*)**.** (1) *If $x \in P_o(A_1^I)$ and $p \in P_o(A_2^J)$, then* $D_f[I, J] = \min\{AS_1, AB_1, AB_2, AC_1, AC_2, AA_1, ACo_1\}$.
(2) *If $x \in P_o(A_1^I)$ and $p \in U(A_2^J)$, then* $D_f[I, J] = \min\{AB_1, AB_2, AA_1\}$.

(3) *If* $x \in U(A_1^I)$ *and* $p \in P_o(A_2^J)$, *then* $D_f[I,J] = \min\{AC_1, AC_2, ACo_1\}$.
(4) *If* $x \in U(A_1^I)$ *and* $p \in U(A_2^J)$, *then* $D_f[I,J] = BS_1$.

Lemma 22 (*Filling up Table* D_l)**.** (1) *If* $x \in P_o(A_1^I)$ *and* $p \in P_o(A_2^J)$, *then*
$D_l[I,J] = \min\{AS_2, AB_3, AB_4, AC_3, AC_4, AA_4, AA_5, AA_6, ACo_4, ACo_5, ACo_6, AD_2, AI_2\}$.
(2) *If* $x \in P_o(A_1^I)$ *and* $p \in U(A_2^J)$, *then*
$D_l[I,J] = \min\{AB_3, AB_4, AA_4, AA_5, AA_6, AD_2, BI_2\}$.
(3) *If* $x \in U(A_1^I)$ *and* $p \in P_o(A_2^J)$, *then*
$D_l[I,J] = \min\{AC_3, AC_4, ACo_4, ACo_5, ACo_6, AI_2, BD_2\}$.
(4) *If* $x \in U(A_1^I)$ *and* $p \in U(A_2^J)$, *then* $D_l[I,J] = \min\{BS_2, BD_2, BI_2\}$.

Lemma 23 (*Filling up Table* D_{fl})**.** (1) *If* $x \in P_o(A_1^I)$ *and* $(x,y) \notin P(A_1^I)$ *and* $p \in P_o(A_2^J)$ *and* $(p,q) \notin P(A_2^J)$, *then*
$D_{fl}[I,J] = \min\{AS_2, AB_3, AB_4, AC_3, AC_4, AA_4, ACo_4\}$.
(2) *If* $x \in P_o(A_1^I)$ *and* $(x,y) \notin P(A_1^I)$ *and* $p \in U(A_2^J)$, *then*
$D_{fl}[I,J] = \min\{AB_3, AB_4, AA_4\}$.
(3) *If* $x \in U(A_1^I)$ *and* $p \in P_o(A_2^J)$ *and* $(p,q) \notin P(A_2^J)$, *then*
$D_{fl}[I,J] = \min\{AC_3, AC_4, ACo_4\}$.
(4) *If* $x \in U(A_1^I)$ *and* $p \in U(A_2^J)$, *then* $D_{fl}[I,J] = BS_2$,
(5) *If* $(x,y) \in P(A_1^I)$ *and* $(p,q) \in P(A_2^J)$, *then* $D_{fl}[I,J] = AS_2$.
(6) *If* $(x,y) \in P(A_1^I)$ *and* $p \in P_o(A_2^J)$ *and* $(p,q) \notin P(A_2^J)$, *then*
$D_{fl}[I,J] = \min\{ABAC_1, ABAC_2, ABACo\}$.
(7) *If* $(x,y) \in P(A_1^I)$ *and* $p \in U(A_2^J)$, *then* $D_{fl}[I,J] = ABBS$.
(8) *If* $x \in P_o(A_1^I)$ *and* $(x,y) \notin P(A_1^I)$ *and* $(p,q) \in P(A_2^J)$, *then*
$D_{fl}[I,J] = \min\{ACAB_1, ACAB_2, ACAA\}$.
(9) *If* $x \in U(A_1^I)$ *and* $(p,q) \in P(A_2^J)$, *then* $D_{fl}[I,J] = ACBS$.

Proof (*Lemma* 20)**.** The principle is similar to the one in the dynamic programming equations to align non-annotated sequences. Indeed, we have to consider the three following configurations between x and p: x and p are aligned together, x is deleted (aligned with $-$) before p, or p is inserted (aligned with $-$) before x. The cases where x or p are aligned with bases but not together are considered through different indexing pairs. The main difference with non-annotated sequence alignment relies on the fact that x and/or p can belong to an arc, that is why we consider four cases: both (resp. none) belong to an arc in case 1 (resp. case 4), only x (resp. p) belongs to an arc in case 2 (resp. case 3).

Let $M_{I,J}$ be an alignment between the arc-annotated sub-sequences A_1^I and A_2^J, defined by $I = (x,y)$ and $J = (p,q)$. We describe below all possible configurations that can be found in $M_{I,J}$ if either x and p are aligned together or one of them is aligned with $-$ before the other. These configurations are illustrated in Fig. 8.

(1) Assume $x \in P_o(A_1^I)$ and $p \in P_o(A_2^J)$. There are three cases depending on the respective alignments of x and p.

(1.a) Assume that x and p are aligned together in $M_{I,J}$. There are again three cases, depending on the configuration of z_1 and/or r_1 in $M_{I,J}$.

(1.a.i) If z_1 is aligned with r_1 then $((x,z_1),(p,r_1)) \in AS(M_{I,J})$ and the inside (resp. outside) of the arc (x,z_1) can only be aligned with the inside (resp. outside) of (p,r_1), which is described by AS_1.
Note that AS_1 also includes the case where if $z_1 = y$ and/or $r_1 = q$, due to the initialization of table D described in Lemma 18. Finally, as (x,z_1) and (p,r_1) are arcs and I and J are indexing pairs of type NMult, the indexing pairs considered in AS_1 belong to NMult.

(1.a.ii) If z_1 is aligned, either with $-$ or with a base of A_2, but after r_1 in $M_{I,J}$, there are again three cases.

(A) z_1 is aligned with a base $r \notin P_e(A_2^J)$, which implies that $(x,z_1) \in AB(M_{I,J})$.
This case is accounted by AB_1, where we record the weight of an arc-breaking for (x,z_1) and align the inside and outside of the arc (x,z_1) with all possible partitions of J into two NMult indexing pairs, as defined by Q_1; we record that both x and z_1 are aligned with bases of A_2 by considering the table D_f in both cases. Note that, by considering D_f in $D_f[(z_1,y),J_2]$, we record that r is aligned with a base, which will be used if $r \in P_o(A_2^J)$ to account for the weight of the edit operation associated with the arc that contains r.

(B) z_1 is aligned with a base $r \in P_e(A_2^J)$, which implies the following two possible cases depending on the base r: either $r \neq r_1$ (then $(x,z_1) \in AB(M_{I,J})$) or $r = r_1$ (then $((x,z_1),(p,r_1)) \in AS(M_{I,J})$).
In both cases, as $M_{I,J}$ belongs to NMult, the bases of A_1^I between x and z_1 (resp. after z_1) can only be aligned with the bases of A_2^J between p and r (resp. after r), and there cannot be an arc in A_2^J with extremities in these two sets of bases. This configuration is considered in AB_2, where adding the weight induced by the fact that $(x,z_1) \in AB(M_{I,J})$ or $((x,z_1),(p,r_1)) \in AS(M_{I,J})$, will be done by calling $D_{fl}[(x,z_1),J_1]$, that records that both bases x and z_1 are aligned with bases of A_2. By construction, all indexing pairs defined by Q_2 are of type NMult.

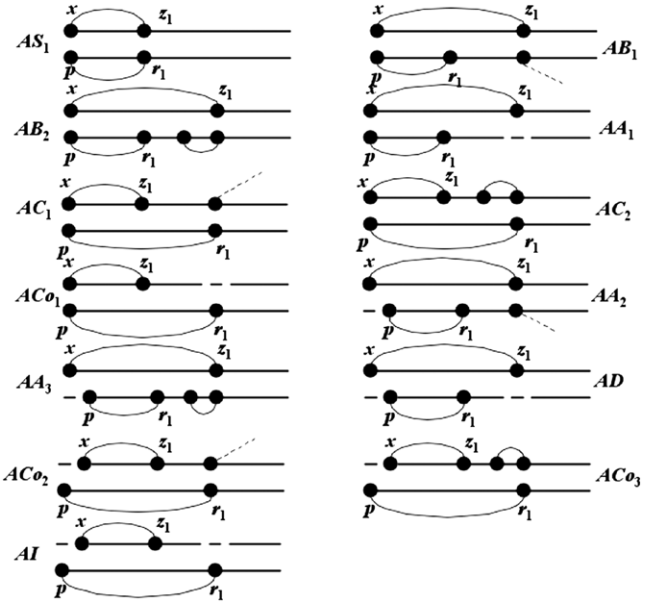

Fig. 8. Illustration of all possible configurations described in the proof of Lemma 20. A dashed line leaving a base indicates that this base can be either unpaired or an extremity of an arc, with the relative position (either on the left or on the right of the considered base) of the other extremity of this arc being indicated by the direction of the dashed line.

(C) z_1 is aligned with $-$ in $M_{I,J}$ and after r_1 (there is a base in A_2^J that is between p and r_1, r_1 included, which is aligned in $M_{I,J}$ with a base of A_1^I that is before z_1), which implies that $(x, z_1) \in AA(M_{I,J})$. This case is considered by AA_1, which records the weight of an arc-altering for (x, z_1), as this arc disappears from the two subsequent indexing pairs of A_1 defining the two considered sub-problems, and ensures that no arc of A_2^J will be crossing with (x, z_1) in $M_{I,J}$, by the definition of Q_1.

(1.a.iii) If z_1 is aligned in $M_{I,J}$ either with a base or with $-$, but before r_1, this case is handled in a symmetrical way by AC_1, AC_2 and ACo_1, where the weight of the operation on the arc (p, r_1) of A_2 is recorded and the weight of the operation on the arc (x, z_1) is postponed to a later sub-problem.

Note that, as in (1.a.i), all the above cases hold when $z_1 = y$ and/or $r_1 = q$, due to the initialization of the tables described in Lemma 18.

(1.b) We now assume that x is aligned with $-$ in $M_{I,J}$, and before p. There are three cases, that we discuss more briefly as the arguments for their correctness are very similar to the previous ones.

(1.b.i) If z_1 is aligned with a base $r \notin P_e(A_2^J)$, then the arc-altering that breaks the arc (x, z_1) is handled by AA_2.

(1.b.ii) If z_1 is aligned with a base $r \in P_e(A_2^J)$, then the arc-altering that breaks the arc (x, z_1) is handled by AA_3.

(1.b.iii) If z_1 is aligned with a base $-$, then the arc-deletion that deletes the arc (x, z_1) is handled by AD.

(1.c) If p is aligned with $-$ in $M_{I,J}$, but before x, this case is handled in a similar way with ACo_2, ACo_3 and AI.

(2) Assume $x \in P_o(A_1^I)$ and $p \in U(A_2^J)$. This case can be deduced from case (1) by not considering any edit operation that assumes p belongs to an arc in A_2^J, namely arc-creation (AC_1 and AC_2), arc-substitution (AS_1), arc-completing (ACo_1, ACo_2 and ACo_3) and arc-inserting (AI).

(3) Assume $x \in U(A_1^I)$ and $p \in P_o(A_2^J)$. This case can be deduced from case (1) by not considering any edit operation that assumes x belongs to an arc in A_1^I, namely arc-breaking (AB_1 and AB_2), arc-substitution (AS_1), arc-altering (AA_1, AA_2 and AA_3) and arc-deleting (AD).

(4) Finally, if $x \in U(A_1^I)$ and $p \in U(A_2^J)$, there are three cases, as in the classical string alignment problem: x and p are aligned together in $M_{I,J}$ (BS_1), x is aligned with $-$ before p (BD), and p is aligned with $-$ before x (BI). □

Proof (*Lemma* 20)**.** The proof follows directly from the proof of Lemma 20 by considering only the cases where x and p are aligned together. □

Proof (*Lemma* 22)**.** The proof is similar to the proof of Lemma 20 but here we impose that y and q are aligned together. This forbids, in some cases, that one of the indexing pairs of a partition of the current indexing pair is empty, which is handled by Q_3. In the case where $z_1 = y$ and/or $r_1 = q$, the initialization of tables D_l and D_{fl} with ∞ when exactly one of the two indexing pairs is empty ensures that y and q are aligned together. □

Proof (*Lemma* 23)**.** Formulas 1–4 correspond to the cases where neither (x, y) is an arc of A_1^I nor is (p, q) an arc of A_2^J. They follow from Lemma 22 where we keep only the configurations where x and p are aligned together.

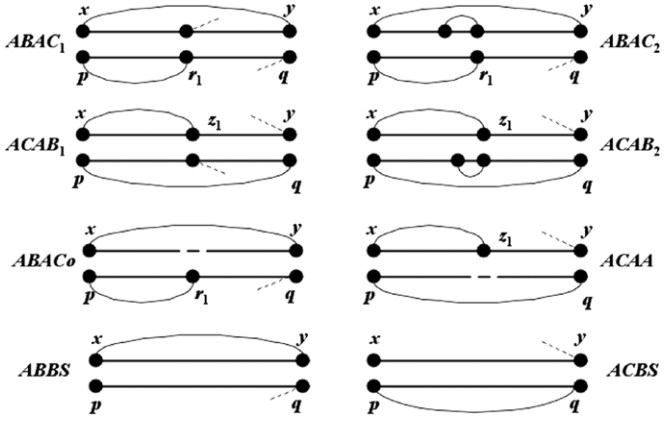

Fig. 9. Illustration of some possible configurations described in the proof of Lemma 23.

Formulas 5–9 correspond to the cases where either only (x, y) is an arc (formulas 6 and 7), or only (p, q) is an arc (formulas 8 and 9), or both are arcs (formula 5). They are illustrated (but formula 5 that corresponds to AS_2) in Fig. 9.

The general principle for these formulas is that as one of the indexing pairs is an arc, and both its bases are mapped, then we need to account for at least one arc operation (an arc-breaking for (x, y) or an arc-creation for (p, q)), and possibly another operation on one of the bases x and p or on the arc that contains it. Q_4 and Q_5 correspond to Q_1 and Q_2 when we allow an arc between the two indexing pairs resulting from a partition, composed of the two extremities of this indexing pair. From these points, the proofs for formulas 5–9 are similar to the previous proofs. □

Before describing the complete algorithm for the computation of the distance between two arc-annotated sequences A_1 and A_2, we study the set of indexing pairs that need to be considered. Given an arc-annotated sequence A of length n, we denote $H(A)$ the set of indexing pairs of A of type NMult such that, either $y = n$, or y is the end of an arc or y is a base just before the end of an arc:

$$H(A) = \{(x, y) \in I(A) \mid y = n \text{ or } \exists\, w \leq x \text{ s.t. } (w, y) \in P(A) \text{ or } (w, y+1) \in P(A)\}.$$

Lemma 24 (*Indexing Pairs to be Considered*)**.** *Let (I, J) be a couple of* NMult *indexing pairs of two arc-annotated sequences A_1 and A_2, that belong to $H(A_1) \times I(A_2) \cup I(A_1) \times H(A_2)$. Then, in Lemma 20, every couple of indexing pairs (I', J') that is involved in the computation of $D[I, J]$ belongs to $H(A_1) \times I(A_2) \cup I(A_1) \times H(A_2)$.*

Proof. We first consider the couples of indexing pairs that are immediately required to compute $D[I, J]$. It is easy to see, case by case and from the definition of Q_1 and Q_2, that they all belong to $H(A_1) \times I(A_2) \cup I(A_1) \times H(A_2)$. The fact that this property holds for the later required indexing pairs follows by induction and from the fact that D_l, D_f and D_{fl} are computed using dynamic programming equations that are restrictions of the ones defined in Lemma 20. □

We now present the main algorithm, that proves Theorem 14.

Proof (*Theorem* 14)**.** The proof relies on two points: (1) The algorithm computes the weight of an optimal NMult-alignment between A_1 and A_2, and (2) the algorithm runs with an $O(n^4)$ time complexity and requires $O(n^3)$ space.

The validity of the algorithm, i.e. the facts that it fills the cells of the tables D, D_f, D_l and D_{fl} according to Definition 17, follows from three points.

- Lemmas 18, 20, 21, 22 and 23.
- The couples of indexing pairs which need to be considered in the dynamic programming algorithm are those of $H(A_1) \times I(A_2) \cup I(A_1) \times H(A_2)$ (Lemma 24).
- The fact that indexing pairs are considered in increasing order of length, and that $D_{fl}[I, J]$ is computed before $D_f[I, J]$ and $D_l[I, J]$ that are themselves computed before $D[I, J]$ (see Fig. 10). .

We now address the complexity of the algorithm. First, a nested arc-annotated sequence A of length n has $O(n^2)$ indexing pairs in $I(A)$ and $O(n)$ indexing pairs in $H(A)$. Hence, from Lemma 24, the four dynamic programming tables require a space in $O(n^3)$. Regarding the worst-case time complexity, the only point which needs to be addressed is that some dynamic programming equations consider a set of partitions of the current indexing pair into two indexing pairs. However, by the definition of the partition of an indexing pair I into two indexing pairs I_1 and I_2, both I_1 and I_2, if non-empty, share an endpoint with I. The time complexity of the algorithm is then bounded by:

$$\left(|H(A_1)| \sum_{J \in I(A_2)} |J|\right) + \left(|I(A_2)| \sum_{I \in H(A_1)} |I|\right) + \left(|I(A_1)| \sum_{J \in H(A_2)} |J|\right) + \left(|H(A_2)| \sum_{I \in I(A_1)} |I|\right)$$

which is asymptotically equivalent to $O(n_1 \times n_2^3 + n_1^2 \times n_2^2 + n_1^2 \times n_2^2 + n_1^3 \times n_2)$ or equivalently to $O(n^4)$. □

Algorithm : $ALIGN(A_1 = (S_1, P_1), A_2 = (S_2, P_2))$
Begin
Initialize tables D, D_f, D_l and D_{fl} using Lemma 18
For $i = 0 \rightarrow |S_1| - 1$ **Do**
For $x = 1 \rightarrow |S_1|$ **Do**
$y = x + i$
If$(x, y) \in I(A_1)$ **Then**
For $j = 0 \rightarrow |S_2| - 1$ **Do**
For $p = 1 \rightarrow |S_2|$ **Do**
$q = p + j$
If$(p, q) \in I(A_2)$ **Then**
If$(x, y) \in H(A_2)$ or $(p, q) \in H(A_2)$ **Then**
$I = (x, y),\ J = (p, q)$
Compute $D_{fl}[I, J]$ using Lemma 23
Compute $D_l[I, J]$ using Lemma 22
Compute $D_f[I, J]$ using Lemma 21
Compute $D[I, J]$ using Lemma 20
Output: $D[(1, |S_1|), (1, |S_2|)]$

Algorithm 1. Algorithm to compute the weight of an optimal NMult-alignment between two nested arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$.

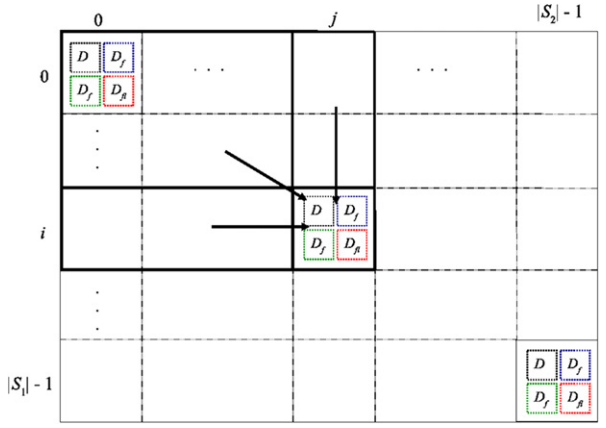

Fig. 10. Illustration of the algorithm as a matrix with $|S_1|$ rows and $|S_2|$ columns. Each cell (i, j) of the matrix represents a step of the algorithm consisting in the computation of the values $D(I, J)$, $D_l(I, J)$, $D_f(I, J)$ and $D_{fl}(I, J)$ for all pairs $(I, J) \in H(A_1) \times I(A_2) \cup I(A_1) \times H(A_2)$ where I is of length $i + 1$, and J is of length $j + 1$. The steps that should be completed prior to step (i, j) are steps (k, l) where $k \leq i$ and $l \leq j$. Thus, the steps of the algorithm are completed in a row-major order: (k, l) precedes (i, j) if $k < i$ or, $k = i$ and $l < j$.

4. Conclusion

We proposed an extension of the hierarchy of arc-annotated sequences, introducing a new class named NMult. Based on this extension and on the arc-annotated sequence alignment framework introduced in [3,5], we introduced new alignment problems. The polynomial time and space algorithm we propose for computing the NMult-alignment distance between two nested arc-annotated sequences solves the most general known tractable problem in the extended hierarchy of alignment problems, when considering all the edit operations introduced in [15] and a complete weight scheme. This extends previous results about Align(Nest, Nest; Nest) [3,5]. As far as we know, the only other alignment algorithm that considered an NMult super-sequence did not consider all edit operations and a complete weight scheme [13].

From an applied point of view, it remains to see if the tractability of Align(Nest, Nest; NMult) has a significant impact on RNA secondary structure alignments. Preliminary results on RNA structures retrieved from the database RFAM [10] showed few examples where the optimal NMult-alignment was different from the optimal Nested-alignment.

From a theoretical point of view, the issue of weighting schemes is fundamental. The hardness of ALIGN(Nest, Nest; Cros) was proved in [4,19], in the case of a weight scheme that prevents to use the arc-breaking operation. It then remains open to extend this hardness result to the case of a complete weight scheme. More generally, no hardness proof for the comparison of arc-annotated sequences using edit operation do rely on a complete weighing scheme, which suggests that some more general tractable problems could be identified with a complete set of edit operations.

Acknowledgements

This work was supported by a grant ANR-06-BLANC-0045 (BRASERO) to A.O. and grant from NSERC and SFU to C.C.

References

[1] R. Backofen, S. Siebert, Fast detection of common sequence structure patterns in RNAs, Journal of Discrete Algorithms 5 (2) (2007) 212–228.
[2] R. Backofen, S. Will, Local sequence-structure motifs in RNA, Journal of Bioinformatics and Computational Biology 2 (4) (2004) 681–698.
[3] G. Blin, A. Denise, S. Dulucq, C. Herrbach, H. Touzet, Alignment of RNA structures, IEEE/ACM Transactions on Computational Biology and Bioinformatics 7 (2) (2010) 309–322.
[4] G. Blin, G. Fertin, I. Rusu, C. Sinoquet, Extending hardness of RNA secondary structure comparison, in: B. Chen, M. Paterson, G. Zhang (Eds.), Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, First International Symposium, ESCAPE 2007, Hangzhou, China, April 7–9, 2007, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 4614, Springer, 2007, pp. 140–151.
[5] G. Blin, H. Touzet, How to compare arc-annotated sequences: the alignment hierarchy, in: F. Crestani, P. Ferragina, M. Sanderson (Eds.), String Processing and Information Retrieval, 13th International Conference, SPIRE 2006, Glasgow, UK, October 11–13, 2006, Proceedings, in: Lecture Notes in Computer Science, vol. 4209, Springer, 2006, pp. 291–303.
[6] J. Couzin, Breakthrough of the year: small RNAs make big splash, Science 298 (2002) 2296–2297.
[7] E. Demaine, S. Mozes, B. Rossman, O. Weimann, An optimal decomposition algorithm for tree edit distance, in: L. Arge, C. Cachin, T. Jurdzinski, A. Tarlecki (Eds.), Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9–13, 2007, Proceedings, in: Lecture Notes in Computer Science, vol. 4596, Springer, 2007, pp. 146–157.
[8] S. Dulucq, H. Touzet, Decomposition algorithms for the tree edit distance problem, Journal of Discrete Algorithms 3 (2–4) (2005) 448–471.
[9] P. Evans, Algorithms and complexity for annotated sequences analysis, Ph.D. Thesis, University of Victoria, Canada. 1999.
[10] P. Gardner, J. Daub, J.G. Tate, E.P. Nawrocki, D.L. Kolbe, S. Lindgreen, A.C. Wilkinson, R.D. Finn, S. Griffiths-Jones, S.R. Eddy, A. Bateman, Rfam: updates to the RNA families database, Nucleic Acids Research 37 (Database issue) (2009) D136–D140.
[11] V. Guignon, C. Chauve, S. Hamel, An edit distance between RNA stem-loops, in: M.P. Consens, G. Navarro (Eds.), String Processing and Information Retrieval, 12th International Conference, SPIRE 2005, Buenos Aires, Argentina, November 2–4, 2005, Proceedings, in: Lecture Notes in Computer Science, vol. 3772, Springer, 2005, pp. 335–347.
[12] D. Gusfield, Algorithms on strings, trees and sequences—Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.
[13] M. Höchsmann, T. Töller, R. Giegerich, S. Kurtz, Local similarity in RNA secondary structures, in: 2nd IEEE Computer Society Bioinformatics Conference (CSB 2003), 11–14 August 2003, Stanford, CA, USA, IEEE Computer Society, 2003, pp. 159–168.
[14] I.L. Hofacker, Vienna RNA secondary structure server, Nucleic Acids Research 31 (13) (2003) 3429–3431.
[15] T. Jiang, G.-H. Lin, B. Ma, K. Zhang, A general edit distance between RNA structures, Journal of Computational Biology 9 (2) (2002) 371–388.
[16] T. Jiang, G.-H. Lin, B. Ma, K. Zhang, The longest common subsequence problem for arc-annotated sequences, Journal of Discrete Algorithms 2 (2) (2004) 257–270.
[17] T. Jiang, L. Wang, K. Zhang, Alignment of trees—an alternative to tree edit, Theoretical Computer Science 143 (1) (1995) 137–148.
[18] E.Y. Jin, C.M. Reidys, RNA pseudoknot structures with arc-length >= 3 and stack-length >= sigma, Discrete Applied Mathematics 158 (1) (2010) 25–36.
[19] G. Lin, Z.-Z. Chen, T. Jiang, J. Wen, The longest common subsequence problem for sequences with nested arc annotations, Journal of Computer and System Sciences 65 (3) (2002) 465–480.
[20] A. Ouangraoua, P. Ferraro, S. Dulucq, L. Tichit, Local similarity between quotiented ordered trees, Journal of Discrete Algorithms 5 (1) (2007) 23–35.
[21] J.S. Pedersen, G. Berejano, A. Siepel, K. Rosenbloom, K. Lindblad-Toh, E.S. Lander, J. Kent, W. Miller, D. Haussler, Identification and classification of conserved RNA secondary structures in human genome, PLoS Computational Biology 2 (4) (2006) e33.
[22] S. Siebert, R. Backofen, MARNA: multiple alignment and consensus structure prediction of RNAs based on sequence structure comparisons, Bioinformatics 21 (16) (2005) 3352–3359.
[23] S. Washietl, I.L. Hofacker, M. Lukasser, A. Hüttenhofer, P.F. Stadler, Mapping of conserved RNA secondary structures predicts thousands of functional noncoding RNAs in the human genome, Nature Biotechnology 11 (23) (2005) 1383–1390.
[24] S. Will, K. Reiche, I.L. Hofacker, P.F. Stadler, R. Backofen, Inferring non-coding RNA families and classes by means of genome-scale structure-based clustering, PLoS Computational Biology 3 (4) (2007) e65.
[25] K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, SIAM Journal on Computing 18 (6) (1989) 1245–1262.
[26] K. Zhang, L. Wang, B. Ma, Computing similarity between RNA structures, Theoretical Computer Science 276 (1–2) (2002) 111–132.